Application

For

United States Utility Patent

Title: MULTI-CLIENT TO MULTI-SERVER SIMULATION ENVIRONMENT CONTROL SYSTEM (JULEP)

Inventors:

Robert Beckwith, 50 Voyagers Lane, Ashland, MA 01721
Robert Sanzone, 11 Seneca Dr., Hudson, MA 01749
Mary Albanese, 336 Beacon St., Boston, MA 01545

I hereby certify that this correspondence is being deposited with the U.S. Postal Service as First Class Mail in an envelope addressed to: Assitant Commissioner for Patents, Washington D.C. 20231, on the date indicated below:

Maria P Kovacs

May 8, 2001 — Maria P. Kovacs

Date — Signature

BACKGROUND OF THE INVENTION

1. Field of Invention

The present invention relates generally to simulators of computer or electronic systems.

2. Description of Related Art

A simulation environment consists of one or more client processes and one or more server processes. For simulation to be useful, it must be repeatable, meaning each run of the simulation should produce identical results from identical input. A minimal configuration consists of a single client and a single server. In the minimal configuration, repeatability is obtained by running only one process at a time (i.e., either the client or the server). This simple approach does not work when there exist more than one client or more than one server. The present invention is concerned with creating a simulation environment which produces repeatable results in any configuration.

SUMMARY OF THE INVENTION

An aspect of the present invention is to provide a simulation environment between one or more servers and clients that leads to repeatable outputs.

Another aspect of the present invention is to provide for a simulation environment that performs its function without the necessity for modifying a simulation kernel in any server process.

Yet another object of the invention in one preferred embodiment is to provide for a simulator that introduces another layer between server and client, a control process. All inter-process messages, such as bi-directional messages between server(s) and client(s), are mediated by the control process. Furthermore, the control process is in charge of running the server(s) and client(s) applications.

The multi-client to multi-server simulation environment control system (JULEP™) guarantees that all servers and all clients are synchronized with respect to an artificial time (simulation time) maintained by the control process.

The above described features and many other features and attendant advantages of the present invention will become apparent from a consideration of the following detailed description when considered in conjunction with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

Detailed description of preferred embodiments of the invention will be made with reference to the accompanying drawings.

FIGURE 1 is a software process relationship model for one preferred embodiment of the present invention.

FIGURE 2 is a software process relationship model for another preferred embodiment of the present invention.

FIGURES 3-6 represent a high level software flowchart for the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Disclosed herein is a detailed description of the best presently known mode of carrying out the invention. This description is not to be taken in a limiting sense, but is made merely for the purpose of illustrating the general principles of the invention. The section titles and overall organization of the present detailed description are not intended to limit the present invention.

The software tool may be written in any computer language, preferably C or C++, and run by a general purpose computer system, preferably a computer with ample primary and secondary memory storage, or any specialized hardware or firmware. Depending on the language used to construct and implement the software of the present invention, the software may have any number of classes, functions, subroutines, objects, variables, templates, module(s), lines of code, portions of code and constructs (collectively and generally, and as depicted by the flowcharts herein, "a process step", "step", "process", "block", "block step", "application", "module" or "software module") to carry out the invention in successive stages as described and taught herein, and may be either a standalone software application, or employed inside of or called by another software application. The software process or software module may be constructed so that one portion of code in the application performs a plurality of functions, as for instance in Object Oriented programming (e.g., an overloaded process). The converse is also true in that a plurality of software modules or process steps may be constructed to perform the function of a single process step described herein, without loss of generality for the present invention. At any stage of the process step of the present invention, intermediate values, variables and data may be stored for later use by the program

FIG. 1 shows a software process relationship model for one preferred embodiment of the present invention. Various entities are shown by oval or circular shapes, representative of both hardware (such as processor, primary and secondary memories, I/O such as keyboard, monitor and the like cooperating with the processor) and software residing in memory and operated by the processor. A control process (C.P.), 110, acts as an interface or intermediate layer between one or more servers 120 and one or more clients 130. Thus rather than the servers interacting directly with the clients, they interact first with the control process, performing the role of a traffic coordinator, which then redirects interprocess messages (or generally, data) to the clients. Likewise the control process 110 interfaces with data received from clients 130, and redirects the data to particular servers assigned to the clients. Furthermore, the control process has the ability to stop and start the execution of a client process and a server process, e.g., to pause the running of the client and server process applications, at select points in time, such as seconds or milliseconds of elapsed time from the start of the simulation. These points in time are called a synchronization points.

By way of example as shown in FIG. 1, during simulation, server 140 (node S_0), server 150 (node S1) and server 160 (node S2) are seen by all clients, such as by clients 170 (node C00), 180 (node C01), 190 (node C10), 1100 (node C20), 1110 (node C21) and 1120 (node C22) as a single entity or system during simulation. All client and server communications pass through and are regulated by control process 110.

It should be further noted that though the simulation program of the present invention runs on a computer system, it is not limited to simply simulating computers, such as a client-server network, but in fact can be used to simulate any system, e.g., Application Specific Integrated Circuits (ASICs), DSL modems, disk drive controllers, graphics processors, network interface adapters and entire communications networks and/or any other mechanical, electromechanical or electronic system.

Though control process 110 in FIG. 1 is shown as a stand alone physical entity, other configurations are possible. For example, as shown in FIG. 2, adopting the same convention of labeling entities as in FIG. 1, the difference is that the control process entity 210, C.P., is now just a software entity residing in one of the servers, such as server 220, S_0. Nevertheless, as in the FIG. 1 embodiment, the control process acts as "traffic cop" or messaging broker between servers and clients, despite the control process being physically resident in one of the servers. In addition, the control process software application may be present as a software module that is resident in the code (e.g., instructions) comprising the software of one of the server applications, such as an external function. Other such configurations are possible without departing from the scope of the present invention.

Typical messages sent and received between client and server (via the control process) include the below message commands:

client messages (messages from clients to servers) include the messages--

- 'peek' (examine a value)
- 'poke' (set a value)
- 'invoke' (instructs the server to perform a task)
- 'create event' (sets up an event to watch for; events are expressions which are continuously evaluated as simulation progresses. When one of these expressions evaluates TRUE, i.e., a non-zero value, the associated event is said to have occurred)
- 'destroy event' (removes an event from the list of events to watch for)
- 'simulate' (simulate until an 'event' or synchronization point)
- 'finish' (exit).

server messages (messages from servers to clients) include the messages--

- 'value' (response to a peek)
- 'acknowledge' (response to the commands poke, invoke and finish; for invoke, if the task being performed by the simulator is expected to return results, the invoking process may use peek to retrieve the results)
- 'event' (an event occurred; this message includes a timestamp)
- 'sync' (a simulation synchronization point has been encountered).

Synchronization (sync) points are periodic pauses in the simulation at periodic points in time. All servers will synchronize at a predetermined interval, determined by the control process. The duration of the interval may vary as simulation progresses, but in general, it is held constant. All servers use the identical interval duration, and the interval duration is maintained by the control process. The interval duration may only be changed at sync points, and if the interval duration is changes, in a preferred embodiment all servers must change to the new interval. It is envisioned, however, that the interval duration may be different for different machines if the application is suitably modified to allow for this contingency.

Turning attention now to FIG. 3, there is shown a high-level flowchart of a typical simulation session of the present invention. First, as shown in FIG. 3, a user starts up the control process (C.P.) or message broker module 310. Next, in block step 320, the control process module reads a configuration file, or text from a command line, which identifies the number of clients and servers, exactly what role each server/client plays, and other initialization factors. In process block step 330, the control process module starts each server and sets the initial synchronization interval. As stated above, sync points, which are the beginning and end points bounding a synchronization interval, are periodic pauses in the simulation, pauses in the execution of instructions by client and server processes. The synchronization interval, which may be thought of as the "granularity" of the artificial time (simulation time) that the simulation system of the present invention runs under, should not be set so small (e.g., smaller in duration than a computer system clock) that nothing can be accomplished during the sync interval by the simulation; nor should it be set excessively large, such as no sync point. In general, however, the operator of the system may set the sync interval at any suitable time, and the sync interval may be adjusted at sync points.

At step 340, each server connects or establishes a session with the control process module and sends the control process module a ready to synchronize message, a notification that the server is ready to simulate. In decision block step 350, the control process module first verifies that all servers have started, and, if so, the control process module starts the client(s) associated with the servers. Box 360 is an entry point for continuing the program; in step block 370 the control process module polls each client for messages. Polling order is not important, but the control process module must poll the clients in a predetermined identical manner each session to ensure repeatability. As the control process module polls clients, the control process accepts messages from clients and forwards them to server processes when the clients issue a 'simulate' message.

In step decision block 380, the client verifies whether all clients have issued such a 'simulate' message before proceeding. Once all clients have issued simulate messages, the control process module instructs the servers to proceed, step 390, which leads to the section of the flowchart labeled User Specified Events (U.S.E.), point box 3100.

At box 410, FIG. 4, the portion of the software program concerned with User Specified Events is disclosed. User Specified Events (U.S.E.) are in general any events to watch for, and may for example be event driven or procedure oriented, and in a preferred embodiment arise in the server. Such User Specified Events are events that reside in the server application process that may, when triggered, such as when an event expression within the server process evaluates to TRUE, instruct the server to act in a specific way towards the client application process(es) that the server process controls. Typically such User Specified Events include interrupt driven events, such as arithmetic overflow, and events such as "Transmit Buffer Empty" (i.e., a buffer that stores data to be transmitted is empty), or "Transmit Buffer Full" (the opposite of "Transmit Buffer Empty", in that the buffer that stores data to be transmitted is full of data), and the like. In decision block step 420, the program checks for the presence of such User Specified Events from the server.

In the simplest case, there are no user specified events and simulation proceeds until all servers reach a synchronization point. Thus in block step 450 simulation proceeds, and in decision block 460 the program checks for the presence of a sync point message from the server. If such a sync point message is present, decision block 4110 checks for whether all servers have reported sync points to the control process module. If all servers have not reported sync points to the control process module, simulation proceeds, as indicated in decision block 4120, which redirects the program to block step 450. If all servers have reported sync points to the control process module, then control is passed to the step 510 (marked "Step Branch Box 510"), as indicated by block step 4150.

Therefore if all servers have reached a sync point, and there are no U.S.E. events and the simulation has not finished, ultimately the control process module will repeat by returning control to the program to step point "BEGIN", step box 360 in FIG. 3. Thus, absent any user specified events, once all servers have reached the sync point, the control process module instructs the servers to proceed again to the next sync point. In this manner the server simulators are kept in pseudo lockstep.

In situations where events are active, that is, the events have been created and will be detected at run-time, simulation proceeds as before, except that when a User Specified Event

(U.S.E.) is encountered, such as indicated in the 'YES' branch of decision block 420 in FIG. 4, an 'event' message (marked EVENT in FIG. 4) is issued by the server for the client. The 'event' message is not sent directly to the client, however, but routed to the control process module. The 'event' message is included with a timestamp of the time, as indicated in block step 440, which is an artificial time (or simulation time) that all server processes are kept on, and maintained by the control process. The timestamp for this artificial time is typically started at an initial time equal to zero elapsed seconds at the start of the simulation program.

The control process module does not deliver this message (or any other event) to any clients until the control process is certain that all servers have simulated at least as far in time as the server simulator sending the 'event' message. The control process knows this either by receiving an event message from the server or a sync message (indicating that the server has reached the next sync point). Thus, for example, assuming no sync point has been detected, as in the 'NO' branch of decision block step 460 in FIG. 4, the program checks in decision block 470 whether all servers – that have not reported a sync point -- have reported with 'event' messages, and, if so (i.e., block 480), proceeds to the point in the program marked 510, Step Branch Box 510 in FIG. 5. If not, the program continues to check for User Specified Events (U.S.E), as per decision block 490, and passing control of the program either back to block step 450 if no U.S.E. are detected, or, if a U.S.E is detected, control is passed to the USE step branch box 430 (block 430, as per block step 4100), and simulation continues.

Turning attention now to FIG. 5, once 'event' messages from all servers not reporting sync points have been received, as from the "Yes" branch of decision block 470, and assuming U.S.E. message(s) exist, as per decision block 520, that is, assuming there is a list of such U.S.E. messages, with the list termed the Pending Event Notification list, the messages received are sorted in server order and then in time order, as indicated in block 530 of FIG. 5.

Server order is a predetermined, consistently applied and repeating pattern or sequence of servers, listed in a sequence or queue by the control process. The queue or sequence can be any sequence of servers, such as, referring to FIG. 1, the sequence $\{S_0, S_1, S_2\}$. Time order is simply the ordering of events in chronological order, using the artificial clock kept by the control process (the artificial clock starts at an initial time equal to zero elapsed seconds when the simulator application program starts). Event messages associated with the earliest time stamp are delivered in client order, as indicated by block 540. Client order can be defined as whatever predetermined arbitrary (but consistently applied) ordering of clients (or queue), with respect to a server(s), that the control process uses to send messages to clients. Thus, clients and servers may be ordered in any manner, but the same ordering method must be used to ensure repeatability. Referring to FIG. 1, such a client order queue may be, by way of example, clients $\{C_{00}, C_{01}\}$ for server S_0, client $\{C_{10}\}$ for server S_1, and clients $\{C_{20}, C_{21}, C_{22}\}$ for server S_2.

In block 550, the client acts on the 'event' message, proceeds with simulation, and when the client is done with the 'event' message, as indicated by decision block 560, proceeds to block 570 where the client sends a 'simulate' message for the server. The 'simulate' message from the client is routed to the control process, which does not forward any 'simulate' message to the servers until all clients with a Pending Event Notification list have received any 'event' message(s) and had the opportunity to act upon them. Thus control is passed back by the program to the decision block step 520, and the process is repeated until there are no more U.S.E messages left in the Pending Event Notification list.

If the Pending Event Notification list is exhausted, the "No" branch of decision block 520 is chosen, control is passed from block 520 to block step 590, which means that the Pending Event Notification list has been processed in its entirety by the control process, all clients have received and acted upon any 'event' messages, and the control process module sends a 'simulate' command message only to those servers that it has not yet received a sync message from, that is, the servers that sent the control process event messages, which would correspond to all the servers in decision block 470 in FIG. 4. Thus, proceeding to the next step at decision block step 5100, and assuming any such servers are present, by traversing the "No" branch of decision block 5100, control of the program is passed back to decision block 470, as indicated by block step 595, and the process is repeated. Note that in the program it may be possible that there are cases where a U.S.E. event will occur at exactly the same time corresponding to a sync point; in the event this happens, all events would be processed before the sync point is recognized, thus events are processed before sync points where there is such a collision.

At some point, there will be no more such servers reporting 'event' messages, and all servers will have reached the next sync point, which should be the same sync point for all servers. At this point, as indicated by the "Yes" branch of decision block 5100, control is passed to the point in the program flowchart marked step FINISH, as indicated by point 5110, corresponding to step point box 610 (step FINISH) in FIG. 6.

Turning now to the finish portion of the software flowchart, FIG. 6, eventually each client will complete and send a 'finish' (exit) message to the control process, as checked for in decision block 620. If so, the program passes control along the "Yes" branch of decision block 620, and the control process will hold each client's 'finish' message until all clients have sent 'finish' messages, and have finished, box 640; otherwise control is passed along the "No" branch of decision block 620 and control is passed back to step BEGIN (box 360 in FIG. 1), as indicated in block 630. Afterwards it is checked to see if all clients of the server are finished, as indicated by the "Yes" branch of decision block 650, otherwise control is passed back to point BEGIN (box 360 in FIG. 1), as indicated by step 660. Assuming that all clients of the server have finished, the control process will send a 'finish' message to each server of the client(s) that have all finished, as indicated by block step 670. Such 'finish' messages include "error" and "warning" counts (i.e., how many errors and warnings have been observed by the process finished during the simulation). The control process sums the errors for all clients finished as well as the warnings for all clients, and passes this information to each server in the finish message the server receives (block 670).

The only valid response to a 'finish' message is either 'finish' or 'acknowledge'. If a server wishes to update the error or warning information from the control process, the server should respond with a finish message, which includes new error and warning counts. Otherwise, the server should simply acknowledge the 'finish' message it receives from the control process. Thus if the control process does not receive a proper 'finish' message, control can be passed back to step BEGIN (box 360 in FIG. 1), as indicated in step 690.

As indicated by block 6100, after the control process has received a suitable response from all servers, the server sending the suitable response to a 'finish' message will terminate the simulation and exit, and an acknowledge (or finish, if the error/warning information has been changed) is then forwarded to each client. As each client receives the acknowledge (or finish), it will also exit. Finally, the control process itself will exit and the program ends (block 6110).

Though the preferred embodiments are disclosed in the present invention, alternative mechanisms may be employed without departing from the scope of the invention. It is to be understood that while the invention has been described above in conjunction with preferred specific embodiments, the description and examples are intended to illustrate and not limit the scope of the invention, which is defined by the scope of the appended claims.